## Vehicle Location Estimation Using MAP

A vehicle at true position $[x_T, y_T]^{\mathrm{T}}$ in 2-dimensional space is to be localized using distance (range) measurements to $K$ reference (landmark) coordinates $\{[x_1, y_1]^{\mathrm{T}}, \ldots, [x_i, y_i]^{\mathrm{T}}, \ldots, [x_K, y_K]^{\mathrm{T}}\}$. These range measurements are $r_i = d_{Ti} + n_i$ for $i \in \{1, \ldots, K\}$, where $d_{Ti} = \|[x_T, y_T]^{\mathrm{T}} - [x_i, y_i]^{\mathrm{T}}\|$ is the true distance between the vehicle and the $i^{th}$ reference point, and $n_i$ is a zero mean Gaussian distributed measurement noise with known variance $\sigma_i^2$. The noise in each measurement is independent from the others.

Assume that we have the following prior knowledge regarding the position of the vehicle:

$$p\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = (2\pi\sigma_x\sigma_y)^{-1} e^{-\frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}} \tag{1}$$

where $[x, y]^{\mathrm{T}}$ indicates a candidate position under consideration.

**Express the optimization problem** that needs to be solved to determine the MAP estimate of the vehicle position. Simplify the objective function so that the exponentials and additive/multiplicative terms that do not impact the determination of the MAP estimate $[x_{MAP}, y_{MAP}]^{\mathrm{T}}$ are removed appropriately from the objective function for computational savings when evaluating the objective.

**Implement the following as computer code:** Set the true vehicle location to be inside the circle with unit radious centered at the origin. For each $K \in \{1, 2, 3, 4\}$ repeat the following.

Place evenly spaced $K$ landmarks on a circle with unit radius centered at the origin. Set measurement noise standard deviation to 0.3 for all range measurements. Generate $K$ range measurements according to the model specified above (if a range measurement turns out to be negative, reject it and resample; all range measurements need to be nonnegative).

Plot the equilevel contours of the MAP estimation objective for the range of horizontal and vertical coordinates from $-2$ to 2; superimpose the true location of the vehicle on these equilevel contours (e.g. use a $+$ mark), as well as the landmark locations (e.g. use a $o$ mark for each one).

Provide plots of the MAP objective function contours for each value of $K$. When preparing your final contour plots for different $K$ values, make sure to plot contours at the same function value across each of the different contour plots for easy visual comparison of the MAP objective landscapes. *Suggestion:* For values of $\sigma_x$ and $\sigma_y$, you could use values around 0.25 and perhaps make them equal to each other. Note that your choice of these indicates how confident the prior is about the origin as the location.

Supplement your plots with a brief description of how your code works. Comment on the behavior of the MAP estimate of position (visually assessed from the contour plots; roughly center of the innermost contour) relative to the true position. Does the MAP estimate get closer to the true position as $K$ increases? Doe is get more certain? Explain how your contours justify your conclusions.

*Note: The additive Gaussian distributed noise used in this question is likely not appropriate for a proper distance sensor, since it could lead to negative measurements. However, in this question, we will ignore this issue and proceed with this noise model for illustration. In practice, a multiplicative log-normal distributed noise may be more appropriate than an additive normal distributed noise depending on the measurement mechanism.*

*3. The objective is to find the $[x,y]^T$ coordinate position with the highest probability given the prior distribution of as well as the range measurements from each of the K reference coordinates.

$$\begin{bmatrix} x_{MAP} \\ y_{MAP} \end{bmatrix} = \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \; p\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| r_1, \dots r_K\right)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \left((2\pi\sigma_x\sigma_y)^{-1} e^{-\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}}\right) \prod_{i=1}^{K} p\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| r_i\right)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} \ln\left((2\pi\sigma_x\sigma_y)^{-1}\right) + \ln\left(e^{-\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}}\right) + \sum_{i=1}^{K} \ln p\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| r_i\right)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} -\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln \mathcal{N}(r_i | 0, \sigma_i^2)$$

$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} -\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln\left((2\pi\sigma_i^2)^{-\frac{1}{2}} e^{-\frac{(r_i - d_i)^2}{2\sigma_i^2}}\right)$$

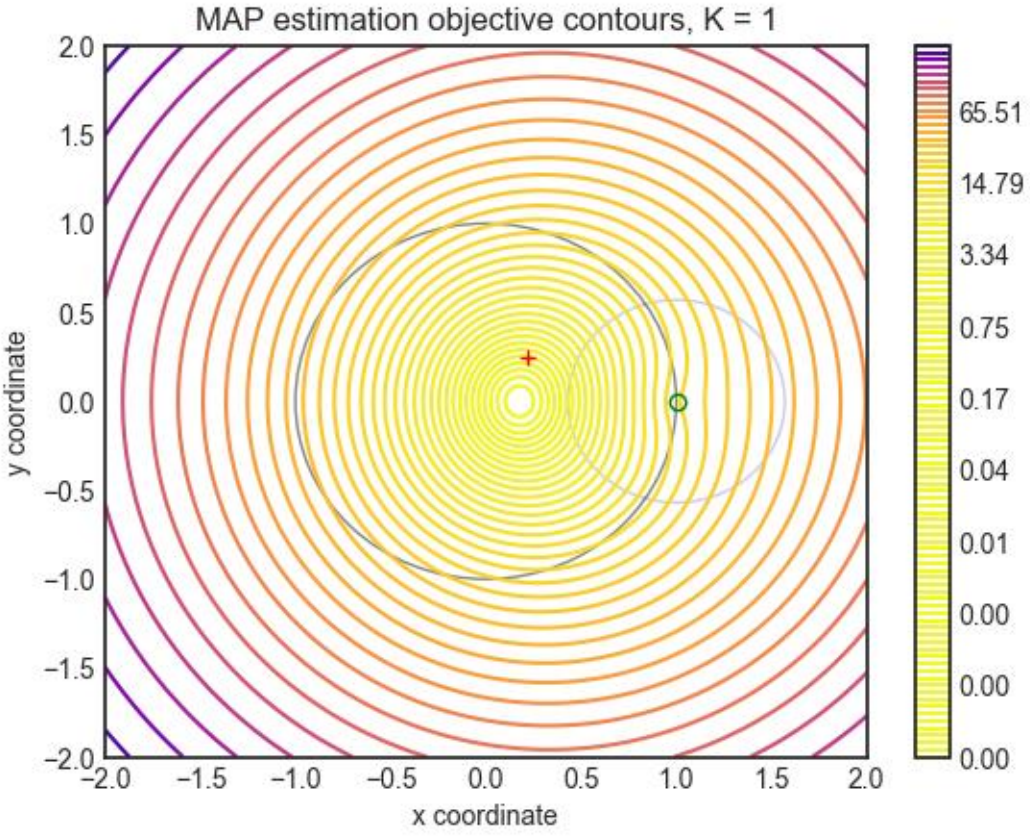
$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} -\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln\left((2\pi\sigma_i^2)^{-\frac{1}{2}}\right) + \ln\left(e^{-\frac{(r_i - d_i)^2}{2\sigma_i^2}}\right)$$

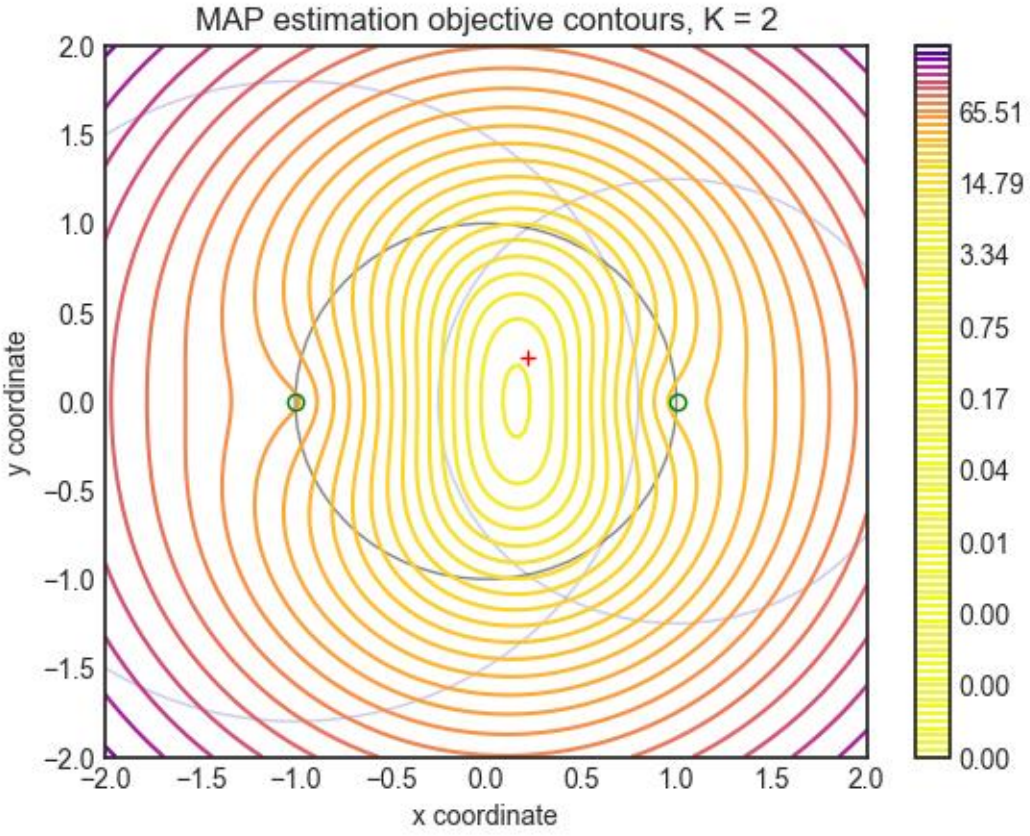
$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmax}} -\frac{1}{2}[x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} -\frac{(r_i - d_i)^2}{2\sigma_i^2}$$

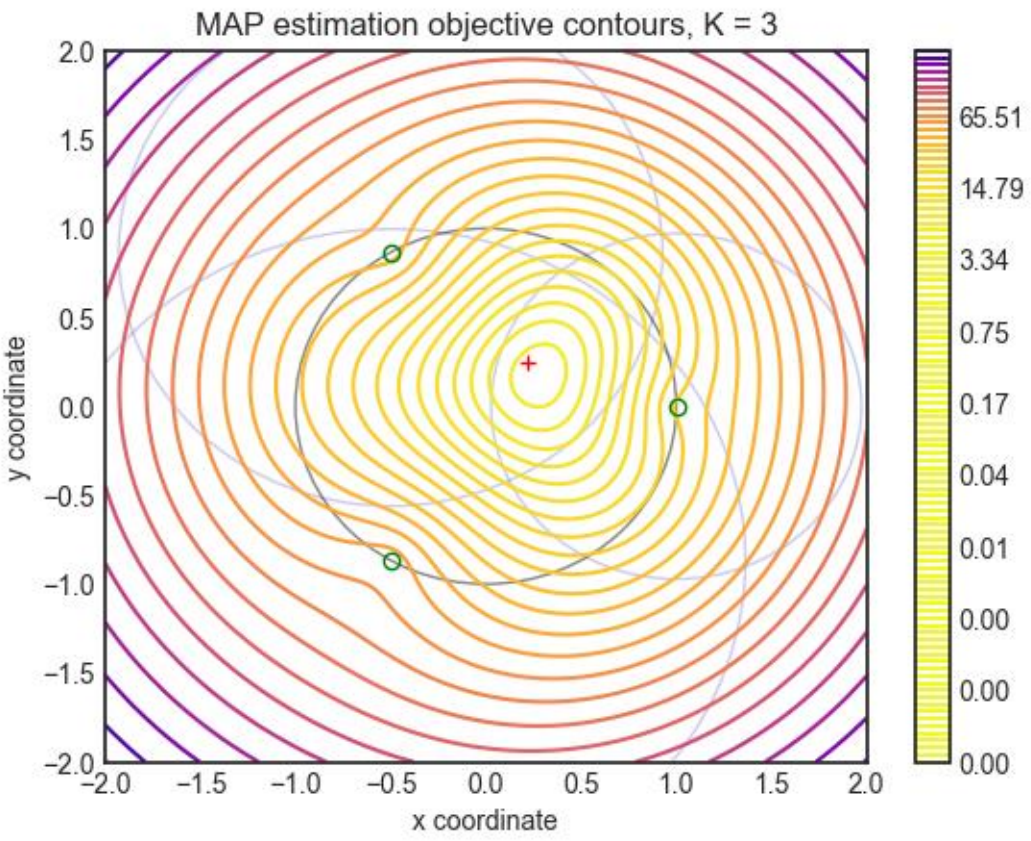
$$= \underset{\begin{bmatrix} x \\ y \end{bmatrix}}{\operatorname{argmin}} [x\ y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \frac{(r_i - d_i)^2}{\sigma_i^2}$$

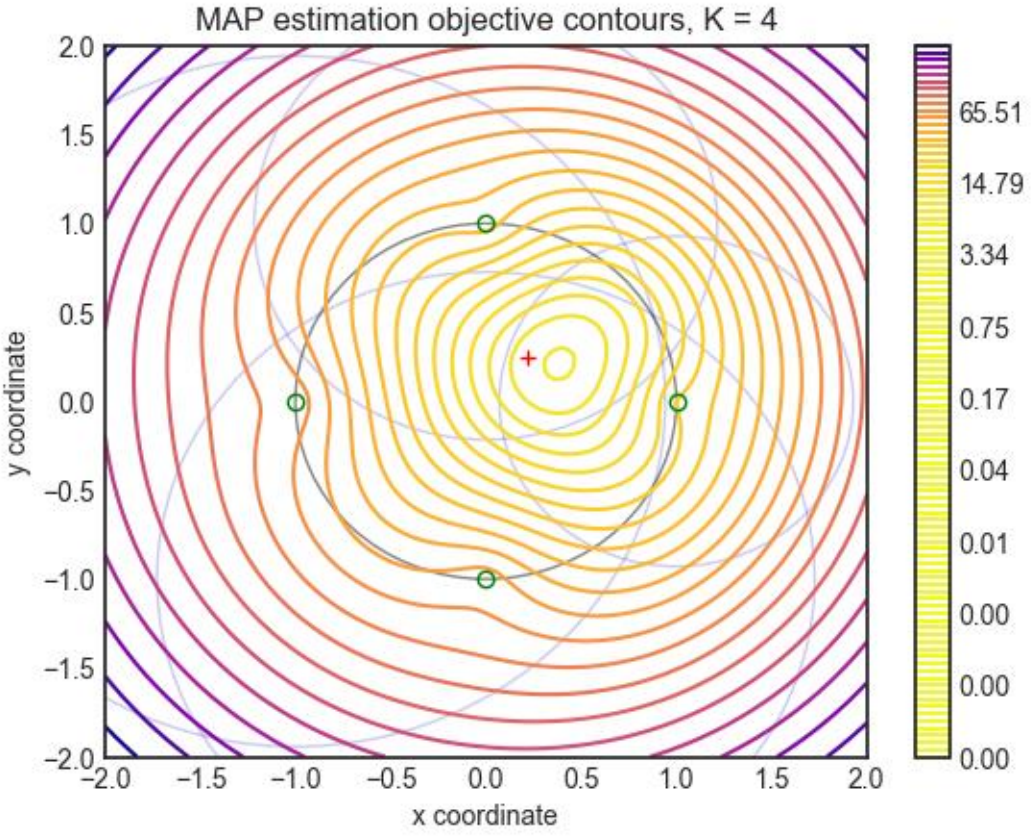
$$d_i = \left\| \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right\|$$

**Results from the code implementation**




MAP estimation objective contours, K = 3
y coordinate
x coordinate
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
65.51
14.79
3.34
0.75
0.17
0.04
0.01
0.00
0.00
0.00


MAP estimation objective contours, K = 4
y coordinate
x coordinate
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
65.51
14.79
3.34
0.75
0.17
0.04
0.01
0.00
0.00
0.00

- The code generates a random coordinate pair within a unit circle at the origin as the true vehicle location.
- For each K, K evenly distributed landmarks around the circle are used to calculate distances from the true vehicle location.
- The measurements are corrupted with additive Gaussian noise and the MAP estimation objective function is calculated for all points on a 128x128 mesh grid.
- The values are plotted as equi-level contours, and the unit circle, true location, landmark locations, and ranges reported by each landmark are also plotted.
- The MAP estimate of position is not very accurate for K < 3 because all estimates are symmetric around the x-axis.
- The accuracy of the estimator increases as K increases.
- The certainty of the estimator can be visualized on the contour graphs by a shrinkage of the area of locations with a high probability.
- The estimator's accuracy can be determined from the contour graph based on the distance from the true location to the point with the lowest contour, roughly around the centre of the innermost contour.
- As K increases, the certainty of the estimator increases.
- For small values of K, it is difficult to notice the shrinkage of the area of locations with a high probability, but following a single contour level as K increases demonstrates this phenomenon well.

Appendix:

```
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

# Function to generate the vehicle's position (inside a unit circle)
def vehicle_position_coordinates(n_samples):
    radius = np.sqrt(np.random.rand(n_samples))
    angle = np.random.rand(n_samples) * 2 * np.pi
    x = radius * np.cos(angle)
    y = radius * np.sin(angle)
    return np.array([x, y])

def get_landmark_positions(k):
    angles = np.linspace(0, 2*np.pi, k+1)[:-1]
    landmarks = np.column_stack((np.cos(angles), np.sin(angles)))
#     print('landmark Values')
#     print(landmarks)
    return landmarks

def get_measurements(true_position, landmark):
    dT = np.linalg.norm(true_position-landmark)
#     print('xy_true')
#     print(xy_true)
#     print('xy_landmark')
#     print(xy_landmark)
```

```
#     print('dTi')
#     print(dTi)
    while True:
        random_noise = np.random.normal(0, SIGMA_I)
        measurement = dT + random_noise
        #return if only the measurement is positive
        if measurement >= 0:
            return measurement

def MAP(measurements, mesh_grid, landmark):
    # The shape of xy is (2, n, m), but it needs to be (n, m, 1, 2).
    mesh_grid = np.expand_dims(np.transpose(mesh_grid, axes=(1, 2, 0)),
axis=len(np.shape(mesh_grid))-1)

    prior = np.matmul(mesh_grid, np.linalg.inv(np.array([[SIGMA_X**2, 0],[0, SIGMA_Y**2]])))
    prior = np.matmul(prior, np.swapaxes(mesh_grid, 2, 3))
    prior = np.squeeze(prior)
    # prior is now of shape (n, m).

    range_sum = 0
#     xy_i = get_landmark_positions(len(measurements))
#     xy_i = np.squeeze(xy_i)
#     print(xy_i)

    for (i, r_i) in enumerate(measurements):
        lm = landmark[i, :]
        d_i = np.linalg.norm(mesh_grid - lm[None, None, None, :], axis=3)
        range_sum += np.squeeze((r_i - d_i)**2 / SIGMA_I**2)
#     print(prior.shape)
#     print(range_sum.shape)
#     print(xy_i.shape)
#     print(xy_i)
    return prior + range_sum


def plotting(xy_true, range_measurements, gridpoints, contour_values, landmark):
    # Then, set up the plot
    plt.style.use('seaborn-white')

    ax = plt.gca()

    unit_circle = plt.Circle((0, 0), 1, color='#888888', fill=False)
    ax.add_artist(unit_circle)

    plt.contour(gridpoints[0], gridpoints[1], contour_values, cmap='plasma_r',
levels=CONTOUR_LEVELS);

#     landmark = get_landmark_positions(len(measurements))
#     landmark = np.squeeze(landmark)
    for (i, r_i) in enumerate(range_measurements):
        (x, y) = landmark[i, :]
```

```
#         print('landmark values')
#         print(landmark[i, :])
        plt.plot((x), (y), 'o', color='g', markerfacecolor='none')
        # I added faint blue circles to demonstrate the range from each landmark
        range_circle = plt.Circle((x, y), r_i, color='#0000bb33', fill=False)
        ax.add_artist(range_circle)

    ax.set_xlabel("x coordinate")
    ax.set_ylabel("y coordinate")
    ax.set_title("MAP estimation objective contours, K = " + str(len(range_measurements)))

    ax.set_xlim((-2, 2))
    ax.set_ylim((-2, 2))
    ax.plot([xy_true[0]], [xy_true[1]], '+', color='r')
    plt.colorbar();
    plt.show()

CONTOUR_LEVELS = np.geomspace(0.0001, 250, 100)
true_position = vehicle_position_coordinates(1)
SIGMA_X = 0.25
SIGMA_Y = 0.25
SIGMA_I = 0.3
# value_set = [1, 2, 3, 4]
value_set = [1, 2, 3, 4]

for k in value_set:
    measurements = []
    #calculating the landmark positions for each k value i.e.. [1, 2, 3, 4]
    #for k = 1 it generates 1 landmark
    #for k = 4 it generates 4 landmarks
    landmark_positions = get_landmark_positions(k)
    # Calculating the measurements for each landmark
    for i in range(len(landmark_positions)):
        measurements.append(get_measurements(true_position, landmark_positions[i]))
    mesh_grid = np.meshgrid(np.linspace(-2, 2, 128), np.linspace(-2, 2, 128))
    contour_values = MAP(measurements, mesh_grid, landmark_positions)
    plotting(true_position, measurements, mesh_grid, contour_values, landmark_positions)
```